NetQuarry, Inc.

Training

600 - Coding Examples

Coding Cookbook

This document provides examples of recommended ways to perform common coding tasks.

Create an extension that will be shared by different mappers

When you create an extension that you want to share between different mappers, you cannot use the TypedMapper derived extension. Instead you use the generic mapper extension.

Declaration of the class

```
using NetQuarry.Data;

namespace CompanyName.Extensions
{
    public class YourSharedClass : NetQuarry.Data.MapperExtensionKernel
    {
        public override void RowBeforeInsert(IMapper sender, EAPEventArgs e)
        {
        }
    }
}
```

Create a typed mapper extension that will be used specifically for a single mapper

When you create a typed mapper extension, it will only be valid for use by the mapper it was based on.

When creating a typed mapper extension you first have to verify that there is a generated TypedMapper class of the mapper. If there is no TypedMapper generated class, then you make sure that one is being generated.

If there is a generated class, you next have to make sure that there is class derived from the generated class. These derived classes are always in the CompanyName.Data namespace, and are nearly always specified in the CompanyName.Data project. You may declare the derived class elsewhere, but it's best to keep them in the same place.

You declare a TypedMapper class as follows

```
namespace CompanyName.Data
{
    public class MyTypedMapper : CompanyName.Data.Generated.my_mapper<MyTypedMapper>
    {
    }
}
```

There doesn't need to be anything else in the class declaration, unless you want to add functionality to the TypedMapper class.

Declaration of the Typed Mapper Extension class

```
using CompanyName.Data;
namespace CompanyName.Extensions
{
    public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
    {
        public override void RowBeforeInsert(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
        {
          CompanyName.Common.Session cs = (CompanyName.Common.Session)sender.Application.Session;
        }
    }
}
```

The recommended way to declare a typed mapper extension is to derive from the CompanyName specific derived class, `CompanyName.Extensions.TypedExtensionBase`, rather than directly from the `NetQuarry.Data.TypedMapperExtension`. Doing so gives you access to additional helper functions, the most useful of these is to obtain a direct reference to the CompanyName Session object. When you create a typed mapper extension, it will only be valid for use by the mapper it was based on.

Declaration of the class

```
using CompanyName.Data;

namespace CompanyName.Extensions
{
    public class MyClass : CompanyName.Extensions.TypedExtensionBase<CompanyName.Data.MyTypedMapper>
    {
        public override void RowBeforeInsert(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
        {
          CompanyName.Common.Session cs = sender.Session;
        }
    }
}
```

You have a basic extension but you want to take advantage of the features of a TypedMapper.

With a regular extension based on MapperExtensionKernel, you may want to make use of TypedMappers in the code. Also in a typed mapper extension you might want to refer to another mapper as a typed mapper in your code.

Let's say the first example is that your basic extension is a mapper extension only for people.
The mapper is called people, the TypedMapper is called PeopleTM.
To convert the basic mapper object to a TypedMapper object, you use the Attach method

```
using NetQuarry.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class YourSharedClass : NetQuarry.Data.MapperExtensionKernel
  {
    public override void RowBeforeInsert(IMapper sender, EAPEventArgs e)
    {
      //--- attach TypedMapper to the existing mapper instance
      using (CompanyName.Data.PeopleTM p1 = CompanyName.Data.PeopleTM.Attach(sender))
      {
        string pk;
        pk = p1.people_id;
      }
      //--- create a new instance of TypedMapper to insert new records
      using (PeopleTM p2 = PeopleTM.OpenNew(this.Application))
      {
        bool loopFinished = false;
        while (loopFinished != false)
        {
          string pk;
          p2.display_name = "Smith, John";
          p2.Save();
          p2.MoveNew();
         //--- this will loop endlessly until you break, or set the loopFinished to true.
        }
        p2.Close();
      }
      //--- create a new instance of TypedMapper to read records, or update records
      using (PeopleTM p3 = PeopleTM.OpenReader(this.Application, sender.Filter, 0, MapperAttrs.NoRowRequery))
      {
        string pk;
        while (p3.MoveNext())
        {
          pk = p3.people_id;
          p3.display_name = string.Format("{0}({1})", p3.display_name, pk);
          p3.Save();
        }
        p3.Close();
      }
    }
  }
}
```

Create a SQL statement to insert, update or delete records

You may want to manually access the database directly in your code without having to perform the task through a mapper. There are a number of ways to do this, but the way in which the code is constructed here is the recommended way to perform these functions.

Write the SQL statement directly

```
using NetQuarry.Data;
using System.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class YourSharedClass : NetQuarry.Data.MapperExtensionKernel
  {
    public override void RowBeforeInsert(IMapper sender, EAPEventArgs e)
    {
      //--- very simple way
      using (CompanyName.Data.PeopleTM p1 = CompanyName.Data.PeopleTM.Attach(sender))
      {
        string pk;
        pk = p1.people_id;
        string newName = string.Format("{0}({1})", p1.display_name, pk);
        string sql = string.Format(@"UPDATE people SET display_name = {0} WHERE people_id = {1}",
                       EAPUtil.AnsiQuote(newName),
                       EAPUtil.AnsiQuote(p1.people_id));

        sender.Database.Execute(sql);
      }

      //--- when the filter value is the current value of a field on a mapper.
      using (CompanyName.Data.PeopleTM p2 = CompanyName.Data.PeopleTM.Attach(sender))
      {
        string pk;
        pk = p2.people_id;
        string newName = string.Format("{0}({1})", p2.display_name, pk);
        string sql = string.Format(@"UPDATE people SET display_name = {0} WHERE {1}",
                       EAPUtil.AnsiQuote(newName),
                       p2.Fields.people_id.BuildFilter());

        sender.Database.Execute(sql);
      }

      //--- when the filter values referred to are not strings (e.g. date)
      //--- EAPUtil.SQLString() is preferred as empty values are converted to NULL SQL
      using (CompanyName.Data.PeopleTM p3 = CompanyName.Data.PeopleTM.Attach(sender))
      {
        string pk;
        pk = p3.people_id;
        string newName = string.Format("{0}({1})", p3.display_name, pk);
        string sql = string.Format(@"UPDATE people SET display_name = {0},
                                                       date_updated = {1} WHERE {2}",
        EAPUtil.AnsiQuote(newName),
        EAPUtil.SQLString(DateTime.Now, OleDb.OleDbType.DBTimeStamp,sender.Database.DBMSType())),
        p3.Fields.people_id.BuildFilter());

        sender.Database.Execute(sql);
      }
    }
  }
}
```

Create a SQL statement to insert, update or delete records

You may want to manually access the database directly in your code without having to perform the task through a mapper. There are a number of ways to do this, but the way in which the code is constructed here is the recommended way to perform these functions.

Use the NetQuarry helper class SQLInserter, SQLUpdater, SQLDeleter

```
using NetQuarry;
using NetQuarry.Data;
using System.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class YourSharedClass : NetQuarry.Data.MapperExtensionKernel
  {
    public override void RowBeforeInsert(IMapper sender, EAPEventArgs e)
    {
      //---
      using (CompanyName.Data.PeopleTM p1 = CompanyName.Data.PeopleTM.Attach(sender))
      {
        string pk;
        pk = p1.people_id;
        string newName = string.Format("{0}({1})", p1.display_name, pk);
        SQLUpdater su = new SQLUpdater("people", p1.Fields.people_id.BuildFilter());
        su.AddColumn("display_name", newName, OleDb.OleDbType.VarChar);
        su.AddColumn("date_updated", DateTime.Now, OleDb.OleDbType.DBTimeStamp);
        su.Execute(sender.Database, SQLHelperFlags.RowLock, "People.RowBeforeInsert");

        sender.Database.Execute(sql);
      }
    }
  }
}
```

Responding to Menu Commands in code.

You would set up commands in meta data for users to perform actions on a single record, selected records, or all records. The number of records acted upon depends on what the developer allows, and what the actions the user takes. The developer controls how the actins must be performed by setting, or not setting the CommandAttributes RequireOneSelection, or RequireSelection. RequireOneSelection means that the action can only be performed on a singly selected record. That is one record checked on the list view, or a detail record. RequireSelection means that the action can only be performed on one ore more selected records. That is one or more records checked in the list view, or a detail record. If neither of these attributes is selected, then the action can be performed with no records selected, meaning all records in the list view matching the filter criteria or the detail record.

To handle the command in your extension, the basic approach is to

1. Identify what records meet the criteria for the command.
2. Create a new mapper on which to perform the actions (never the mapper firing the event).
3. Filter the new mapper based on the records selected.
4. Perform the actions related to the command on each record of the new mapper
5. Close the new mapper when complete.

```
using NetQuarry;
using NetQuarry.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class MyExt : CompanyName.Extensions.TypedExtensionBase<CompanyName.Data.MyTM>
  {
    public override void MapperCommand(CompanyName.Data.MyTM sender, EAPCommandEventArgs e)
    {
      switch (e.CommandName)
      {
        case "doSomethingFun":
          DoSomethingFun(sender);
          break;
      }
    }
  }

  private void DoSomethingFun(CompanyName.Data.MyTM sender)
  {
//--- Identify the records to be acted upon
//--- If there are selected keys, then some rows were selected by user
//--- If there are no selected keys, then just use the filter that is on the mappper
    ArrayList arrKeys = sender.Mapper.SelectedKeys();
    string rowFilter = (arrKeys.Count > 0) ? base.GetMapperKeyFilter(arrKeys, "people_id") :
                this.Mapper.Filters.GetFilter(GetFilterFlags.IncludeAllTypes);
    //--- The use of NoRowRequery attribute is required when looping through a mapper and saving
    //--- records in this way. Due to the requery after save, the filter set up programmatically has
    //--- been lost and replaced by a filter on the row key updated.
    //--- Therefore the rows to be
    //--- Additionally, because it cause a requery on the
    using (People p = People.OpenReader(this.Application, rowFilter, 0, MapperAttrs.NoRowRequery))
    {
      while (p.MoveNext())
      {
        //--- Do something interesting to each record of people.

        p.Save();
      }
      p.Close();
    }
  }
}
```

When you can't specify the correct Navigate metadata on a button, do it in code. (Handling button click)

For the most part you can set up buttons and links to perform all the navigation tasks you require. However, it is sometimes necessary for navigation links to act differently depending on the data in the mapper. This is required for when supplying field references in navigation meta data is not sufficient.

Let's say you want to navigate to a different version of a detail depending on some value in a mapper. The navigate primary key is the same for all types of navigation, the page is dependent on the value.

For all of these cases, you want to use the Button cell type. That type of field results in a ButonClick event being fired on any extensions attached to the mapper. The Link cell type does not fire an event. If you prefer that the field looks like a link instead of a button, then you would set the ButtonType field property depending on whether you want a link on the list view and or detail.

```
using NetQuarry;
using NetQuarry.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class MyExt : CompanyName.Extensions.TypedExtensionBase<CompanyName.Data.MyTM>
  {
    public override void FieldButtonClick(IField sender, EAPEventArgs e)
    {
      switch (sender.Key)
      {
        case "btn_view_something":
        {
           //--- Assuming the field is on a people mapper, you can attach to typed mapper
           using (People p = People.Attach(sender.Mapper))
           {
             //---User navigates to their detail, otherwise it's a client user and go to their detail
             string target = sender.role_type.Left(9) == "Corporate" ? "people!corpdetail" : "people!clientdetail";
             string pk = p.people_id;
             //---This is optional, but what you get in doing this is some contextual information available
             //--- on the target page about where the navigation came from.  Sometimes that is useful.
             string qs = string.Format("parkey={0}&parmop={1}&parmap={2}&parrk={3}",
                                       sender.Mapper.RowKey,
                                       sender.Mapper.MOP,
                                       sender.Mapper.Key,
                                       sender.Mapper.RowKey);

             this.Application.Navigate(target, pk, qs, "nav");
           }
          break;
        }
        default:
          break;
      }
    }
  }
}
```

How to access the query parameters in an extension

Occasionally you will want to know from which location a particular page was navigated to. Or, you want to pass some contextual information from one page to another. In either of these cases you need to know how to access the query string parameters.

In the second example, we mentioned how you would pass contextual data from one page to another. That could be achieved by navigating to the page using code similar to the example given at “When you can’t specify the correct Navigate metadata on a button, do it in code”.

Before you write code to extract information from the query parameters, you have to add a reference to System.Web in your project

```
using NetQuarry.Data;
using System.Web;

namespace CompanyName.Extensions
{
  public class YourSharedClass : NetQuarry.Data.MapperExtensionKernel
  {
    public override void MapperBeforeRequery(IMapper sender, EAPEventArgs e)
    {
       HttpRequest req = HttpContext.Current.Request;
       //--- check if the originator_key parameter has been added to the query string
       string specialParam = req["originator_key"]“;
       //--- always use this method to check if a string is empty.
       if (!string.IsNullOrEmpty(specialParam))
       {
         //--- special key is detected, so do something
       }
       else
       {
         //--- special key not detected, so do something else
       }
    }
  }
}
```

Hide pages/navigator links

You might want to hide a page from view, or prevent a user from clicking on a link because they are not allowed to.

The majority of situations like this you can simply use meta data permissions on objects. However, you may have user specific rule (preferences for example) where a user is not allowed to see a link or page, even though their role type allows visibility of such a page. Because the visibility of the page/link is based on operational data and not metadata, we have to change the visibility in code after interrogating the appropriate operational data.

The location of this code is critical to the success of the code. You have to remove the object from its collection after the application has loaded the object meta data, but before the collections are interrogated by the application to display to the user. The best location to place this code is in the AfterLoad application event. Typically there is a handler for Application AfterLoad in a startup extension derived from NetQuarry.ApplicationExtensionBase.

```
using NetQuarry;
using NetQuarry.Data;
namespace CompanyName.Extensions
{
  public class YourStartupExt : NetQuarry.ApplicationExtensionBase
  {
      public override void AfterLoad(IAppContext sender, EAPEventArgs e)
      {
         //--- If the Create New Order navigiator link exists, verify the user is
         //--- allowed to create an order by checking their preference.
         //--- Hide the navigator if necessary.
         Navigator nav = null;
         NavTarget nt = null;

         nav = sender.Navigators.Find("new_tree", NavLoadTypes.LoadByName);
         nt = nav.Targets["new order"];
         if (nt != null)
         {
            if (!cs.NewOrder)
            {
               nt.Attributes |= NavTargetAttrs.Hidden;
               DevLog.LogMessage("CompanyName.Extensions.StartupExt", "SetPermissionsBasedOnPrefs", "Disabling
new order creation due to user rules settings", LogMessageLevel.MildWarning);
            }
         }
      }

      //--- convert the generic session to a CompanyName session.
      CompanyName.Common.Session cs = (CompanyName.Common.Session)sender.Session;

      //--- hide the first page of the new order wizard if a client user logs in.
      if (cs.UserType == UserTypes.ClientUser)
      {
        PageInfo pi = sender.PageInfos["order_new!wizard"];
        if (pi != null)
        {
           PageElementInfo pei = pi.Elements["company_select"];
           if (pei != null)
             pei.Attributes |= PageElementAttrs.Disabled;
        }
      }

    }
  }
}
```

Skipping wizard pages

On a wizard with multiple pages, you may want to skip over a page if certain criteria is, or isn't met. If you want this to happen, you must do this in Page Extension.

```
using NetQuarry.Data;
using CompanyName.Common;
namespace CompanyName.PageExts
{
  public class MyWizard : PageExtensionBase
  {
    protected override void WizardNext(object sender, WizardPageEventArgs e)
    {
      //--- You can get to the underlying mapper for the page from the event args
      //--- You shouldn't cast to a typed mapper.Each wizard page can be a different mapper
      //--- NEVER close this mapper.  You simply have a reference to it.
      IMapper map = e.WizardPage.Mapper as IMapper;
      //--- You can identify the name of the page you have just left
      switch (e.WizardPage.PageElementInfo.Name)
      {
         case "page_choose_what_to_do":
           //--- To refer to values entered on the wizard, access the UserData collection
           //--- Get data via INSTANCE:KEYNAME syntax
           //--- on the previous page in the instance "wizard_instance"
           //--- there was a checkbox field with the keyname "view_details"
           bool viewNext = EAPUtil.ToBool(WizardPage.UserData["wizard_instance:view_details"]);

           if (!viewNext)
           {
              //--- We don't want to view the next page
              //--- Tell the wizard what the next page should be
              //--- NEVER refer to the pages directly by numeric index.
              //--- Use the built in functionality to match the page index by name
              e.NextPage = e.Wizard.GetPageNumber("final_summary_page");
           }
           break;
         default:
           break;
      }
    }
  }
}
```

There's no need to code up equivalent code in wizard previous pages. There is a navigation stack that remembers that pages that were actually visited moving forwards through the wizard.

On wizard, create a filter in code and use it on the next page to filter some data.

Let's say you have a multipage wizard where you use the wizard to create some filter criteria. After creating the filter criteria on one page, you want the data on the next page to be filtered by the criteria just created.

This type of approach can be used for any situation where you want wizard entered data to be referred to in the metadata of the same wizard.

The basic idea is to create some filter criteria, then push the filter into the UserData collection.
You have specified in the meta data for the next page of the wizard, that the page should be filtered by this UserData item.

```
using NetQuarry.Data;
using CompanyName.Common;
namespace CompanyName.PageExts
{
  public class MyWizard : PageExtensionBase
  {
    protected override void WizardNext(object sender, WizardPageEventArgs e)
    {
      //--- You can get to the underlying mapper for the page from the event args
      //--- You shouldn't cast to a typed mapper.Each wizard page can be a different mapper
      //--- NEVER close this mapper.  You simply have a reference to it.
      IMapper map = e.WizardPage.Mapper as IMapper;
      //--- You can identify the name of the page you have just left
      switch (e.WizardPage.PageElementInfo.Name)
      {
         case "filter_setup":
           MapperFilters mfs = new MapperFilters();
           string filter = BuildFilterFromMapperValues(map);

           if (filter == string.Empty)
           {
             throw new EAPException("You must enter search criteria");
           }
           else
           {
             mfs.Add("company_id", map.Fields["company_id"].BuildFilter());
             mfs.Add("wizard_filter", filter);
           }

           //--- You can specify any instance:keyname combination to poke the value into UserData
           //--- as long as it's unique for your wizard
           e.WizardPage.UserData["wizard_results:custom_filter"] = mfs.ToString();
           break;
         default:
           break;
      }
    }
  }
}
```

In your wizard meta data, you go to the page element where the results are displayed. Let's say this is a WizardPhantomList type page where there is a Filter property on the page element.

Set the Filter property on that page to [wizard_results.custom_filter]. NOTE: The subtle difference between the way you name the UserData element with a colon, yet refer to that value in meta data with a dot separator.

At runtime, the reference to the instance value is resolved before the results page is rendered.

Change the Finish Target parameters on a wizard based on page values

Normally when you create a wizard, your finish, or cancel action navigation is static. You will always navigate to the same finish target (probably the detail of the thing you created/modified) or cancel target (the thing you came from). However, you may need to dynamically navigate to different locations (for example, chain to other wizards) depending on the values selected by the user.

Typically you would modify the Cancel Action parameters in the WizardCancel event, or modify the Finish Action parameters in the WizardFinish event. There is a limitation however in the WizardFinish event, in that the values from the last page have not yet been transferred from the page to the underlying mapper. If the field which decides the location of navigation is only set on the last page of the mapper, then you have to use the WizardDataExchange event, filtered by the WizDataExchangeType.FinalPageToUserData event argument.

```
using NetQuarry.Data;
using CompanyName.Common;
namespace CompanyName.PageExts
{
  public class MyWizard : PageExtensionBase
  {
    protected override void WizardDataExchange(object sender, WizardDataExchangeArgs e)
    {
      IMapper map = e.WizardPage.Mapper as IMapper;
      if (e.ExchangeType == WizDataExchangeType.FinalPageToUserData)
      {
        //--- any logic test
        if (EAPConvert.ToBool(map.Fields["goto_this_page"].Value))
        {
          e.WizardPage.PageInfo.Properties.Add("FinishAction", "mop_with_pk"); //--- Mop with Item
          e.WizardPage.PageInfo.Properties.Add("FinishTarget", "module!this_page");//--- Target
        }
        else if (EAPConvert.ToBool(map.Fields["goto_that_page"].Value))
        {
          e.WizardPage.PageInfo.Properties.Add("FinishAction", "mop"); //--- Mop
          e.WizardPage.PageInfo.Properties.Add("FinishTarget", "module!that_page");//--- Target
          e.WizardPage.PageInfo.Properties.Add("FinishQueryParams", "pk=[instance.that_id"); //--- QueryParams
                                                                                             // reference to instance value
        }
        else if (EAPConvert.ToBool(map.Fields["goto_new_thing"].Value))
        {
          e.WizardPage.PageInfo.Properties.Add("FinishAction", "mop_new"); //--- New navigation
          e.WizardPage.PageInfo.Properties.Add("FinishTarget", "module!new_thing");//--- Target
          e.WizardPage.PageInfo.Properties.Add("FinishQueryParams", "origmop=[instance.that_id"); //--- QueryParams
        }
        else
        {
          //--- return to sender
          e.WizardPage.PageInfo.Properties.Add("FinishAction", "return"); //--- Mop
        }
      }
    }
  }
}
```

List of Action values required for code

Action Property Name	Action Meaning
"blank"	TargetType.BlankPage
"mop"	TargetType.Mop
"mop_with_pk"	TargetType.MopWithItem
"mop_new"	TargetType.MopNewRecord
"restart"	TargetType.RepeatWizard
"return"	TargetType.RtnToCaller

Checking whether fields are on a mapper

You are likely to come across situations where you have code and meta data that doesn't quite match up. Because of the numerous combinations of Flavors available to developers and flavors applied by the application, it is common to find that a mapper field being referenced in code is excluded from the mapper,

At run time, you will get a Null Pointer Exception because the field object your code references is not there.

You can be defensive about this situation and check to see whether a field exists on the mapper before you actually refer to it,

```
using NetQuarry;
using NetQuarry.Data;
using CompanyName.Data;

namespace CompanyName.Extensions
{
  public class MyClassGeneric : NetQuarry.Data.MapperExtensionKernel
  {
    protected override void RowCurrent(IMapper sender, EAPEventArgs e)
    {
      //--- Method used for on a generic IMapper
      if (sender.Fields.ContainsKey("some_field"))
      {
         //--- field “some_field” was found in the mapper
      }
    }
  }

  public class MyClass : CompanyName.Extensions.TypedExtensionBase<CompanyName.Data.MyTypedMapper>
  {
    protected override void RowCurrent(IMapper sender, EAPEventArgs e)
    {
        //--- Method used for typed mapper
        if (!EAPUtil.IsNullOrBlank(sender.some_field))
        {
           //--- field “some_field” was found in the mapper
        }

        //--- Alternate method
        if (!EAPUtil.IsNullOrBlank(sender.Fields.some_field))
        {
           //--- field “some_field” was found in the mapper
        }

        //--- Generic mapper checking technique on typed mapper
        if (sender.Mapper.Fields.ContainsKey("some_field"))
        {
           //--- field “some_field” was found in the mapper
        }
  }
}
```

Change the content of a picklist programmatically

You may come across occasions where you need to a have a picklist that's dynamically created at runtime. For example, you want a picklist to contain records that have been associated with a specific record. Practical examples are showing a list of cost codes on a timesheet where the cost codes are from the assignment, and not all the cost codes for a particular client.

The basic idea is to

- construct a SQL statement that is appropriate for the picklist
- programmatically set that SQL as the source of the picklist.
- make the replacement at the correct point in the lifecycle of the mapper.

When you know you will always require programmatically replaced picklists, there is no need to associate the field with a picklist in meta data.

The correct point in the lifecycle to replace a picklist is in any Mapper Event handler from MapperAfterLoad to RowCurrent event handlers (inclusive). After RowCurrent event has been fired, the fields are rendered and picklists are queried to display picklists and/or resolve keys.

You can of course replace the picklist any time before row current, but invariably it is only at the RowCurrent event that you have sufficient contextual information to construct the picklist and select the appropriate data. (The mapper is requeried and on a row and you can extract data from the fields of the mapper). Prior to RowCurrent, say MapperBeforeLayout you can get contextual information about the parent of the mapper. If that is all you need to construct a picklist, then you can replace your picklist at that point

```
using CompanyName.Data;
namespace CompanyName.Extensions
{
    public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
    {
        public override void RowCurrent(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
        {
          SetupCostCodePicker(sender);
        }
    }

    private void SetupCostCodePicker(CompanyName.Data.MyTypedMapper sender)
    {
      string sSQL = string.Format("SELECT cost_center_id as pick_key, cost_center_id FROM
                                   assignments_cost_codes WITH(NOLOCK)WHERE display = 1
                                   AND {0} ORDER BY cost_center_id",
                                   sender.Fields.assignment_id.BuildFilter());

        PicklistItemInfo  pl = new PicklistItemInfo("assignment_cost_codes", sSQL,
                                   Database.DatabaseID,
                                   PicklistAttrs.LimitToList,
                                   PicklistType.SQL);

        sender.Fields.cost_center_id.Exec(FieldExecCmds.PicklistReplace, 0, pl);

        //--- Default to first non-blank item in list (0th item is blank).
        fld.DefaultValue = (fld.Picklist.Count > 1) ? fld.Picklist[1].DisplayText : null;
    }
}
```

Change the content of a picklist programmatically

You can also replace a picklist with a pre-defined picklist

```
using CompanyName.Data;
namespace CompanyName.Extensions
{
    public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
    {
        public override void RowCurrent(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
        {
          GetReasonPicklist(sender);
        }
    }

    private void GetReasonPicklist(CompanyName.Data.MyTypedMapper sender)
    {
      PicklistItemInfo pi = sender.Application.Picklists["case_reasons_client"].ItemInfo;

      if (pi != null)
      {
         sender.Fields.case_reason_id.Exec(FieldExecCmds.PicklistReplace, 0, pi);
      }
    }
}
```

Copy values from one instance of a mapper to another

If you have code where you have two identical mappers and you want to transfer the values from one mapper to another, you can perform the copy operation with one line of code instead of writing your own loop.

```
using NetQuarry.Data;

using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
  {
     public override void RowAfterInsert(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
     {
        createPeopleExpenses(sender);
     }
  }
  public void createPeopleExpenses(CompanyName.Data.MyTypedMapper sender)
  {
    using (CcompaniesExpenses ce = CompaniesExpenses.Open(sender.Application,
                                                           sender.Fields.company_id.BuildFilter()))
    {
       using (PeopleExpenses pe = PeopleExpenses.OpenNew(sender.Application))
       {

          //--- semi-colon delimited list of key names to not copy between mappers
          string excludeFromCopy = null;
          //--- semi-colon delimited list of key names that must be copied between mappers
          string includeInCopy = null;
          //--- SetValFlags to control how copy is executed.
          SetValFlags svf = SetValFlags.OnlyIfNullOrBlank;
          pe.Mapper.Exec(MapperExecCmds.CopyValuesFrom, svf, ce.Mapper, includeInCopy, excludeFromCopy);
          pe.people_id = sender.people_id;
          pe.Save();
          pe.Close();
       }
       ce.Close();
    }
  }
}
```

Replace the default subform filter (using RelatedMapperContext)

In normal situations the mechanism that automatically sets up a filter between the parent and subform record is perfectly adequate. That's because for most situations you have a simple single key foreign relationship between the two sets of records. Where the default handling breaks down is when you need to have more complex filtering of the subform records. What is complex about the filtering mey be something as trivial as requiring two filter keys, rather than one.

This example shows how you would override the default subform filtering mechanism.

The basic mechanism is…

- Always handle in the MapperBeforeRequery handler
- Make sure you are only performing these actions on a subform (optional)
- Throw away the default subform filter
- Create a new filter
- Apply the new filter to the mapper.

This first example shows how you can use the ParentContext to change the filter parameters on a subform.

```
using NetQuarry.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
  {
    public override void MapperBeforeRequery(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
    {
      string where = string.Empty;
      //--- The EAPUtil.BitsSet method performs bit testing and is a convenient and
      //--- readable way to test for existence of bits in any set of attributes.
      //--- Here we're making sure to check that we're on a subform displaying a list
      //--- Any other situation and we don't want to mess with filtering.
      if (EAPUtil.BitsSet(sender.Mapper.Flavor, Flavors.Subform) &&
          EAPUtil.BitsSet(sender.Mapper.Flavor, Flavors.ListView))
      {
        //--- Use the ParentContext to get the RelatedMapperContext of the parent
        //--- and attach a TypedMapper to the parent mapper
        //--- The method GetParent() instantiates a live mapper object from the parent info.
        //--- NEVER CLOSE the parent context object.
        using (ParentTM parent = ParentTM.Attach(sender.Mapper.ParentContext.GetParent()))
        {
          //--- Discard the Foreign Key filter from the mapper's filter collection.
          sender.Mapper.Filters.Remove("FK");
          //--- requery the parent.  It uses the parent RowKey as a filter criteria
          parent.Requery();

          //--- cosntruct a new filter
          where = string.Format(@"{0} AND {1}",
                                parent.Fields.order_id.BuildFilter(),
                                parent.Fields.people_id.BuildFilter());

          //--- Add the new filter to the mapper's filter collection
          //--- Give this filter a descriptive name.  It appears in the F8 info.
          sender.Mapper.Filters.Add("FK_override_def_subform", where);
        }
      }
    }
  }
}
```

Show/Hide Menu Commands Programmatically

Menu commands can be programmatically hidden from or displayed to the user when you want to set the visibility dynamically based on a complex set of rules that can't be handled by permissions, or when you want to change the visibility based on version control.

This first example shows how you can use the ParentContext to change the filter parameters on a subform.

```
using NetQuarry.Data;
using CompanyName.Data;
using CompanyName.Common;
namespace CompanyName.Extensions
{
  public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
  {
    public override void RowCurrent(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
    {
      //--- Show a menu item that is currently hidden by default in meta data
      //--- When the version is correct, the menu item will be displayed
      if (sender.Application.Versions["version_controlled_feature"].Number >= 4100)
      {
        Utils.showWorkFlowMenu(sender.Mapper, "cool_new_feature_command");
      }
      //--- Show menus appropriate to the status of the record
      //--- You can use code generated picklist enumerations for comparison and readability.
      if (sender.status_id = (int)CompanyName.Data.Picklists.assignment_status.Enabled)
      {
        //--- hidden by default in meta and visible while assignment is active
        Utils.showWorkFlowMenu(sender.Mapper, "end_assignment");
      }
      else
      {
        //--- enabled by default in meta and hide when assignment complete
        Utils.hideWorkFlowMenu(sender.Mapper, "create_timesheet");
      }
    }
  }
}

namespace CompanyName.Common
{
  public static class Utils
  {
//--- This method is provided here for clarity but should use
    public static void hideWorkFlowMenu(IMapper sender, string commandName)
    {
      if (sender.Commands[commandName] != null)
      {
        sender.Commands[commandName].Attributes |= UICommandAttrs.Hidden;
      }
    }

    public static void showWorkFlowMenu(IMapper sender, string commandName)
    {
      if (sender.Commands[commandName] != null)
      {
        sender.Commands[commandName].Attributes &= ~UICommandAttrs.Hidden;
      }
    }
  }
}
```

Adding Menu Commands Programmatically

Menu commands are typically added in meta data. There are times when you want a command to appear in various locations to provide a standard feature to the user. An example of such a feature is to export a list to Excel.

Instead of adding the command menu to every menu location and then hiding it where it's not needed, we use the Export To Excel mapper extension to add the command to the menu programmatically. So, wherever you want the ability to export data to excel, you simply add the Export to Excel mapper extension to a mapper and you automatically get the associated mapper command.

You always have to add the Mapper Command in the MapperBeforeLayout event handler. After that event is complete, it's too late for the changes to appear on the menu or toolbar.

```
using NetQuarry.Data;
using CompanyName.Data;
using CompanyName.Common;
namespace CompanyName.Extensions
{
  public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
  {
    //--- Declare a const name for your command and then you can refer to it in many places
    //--- Like the MapperCommand event handler!
    private const string SPECIAL_COMMAND = "SPECIAL_COMMAND";
    private const string COMMAND_CAPTION = "COMMAND_CAPTION";
    private const string COMMAND_TOOLTIP = "COMMAND_TOOLTIP";

    public override void MapperBeforeLayout(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
    {
      AddMapperCommand(sender);
    }

    public override void MapperCommand(CompanyName.Data.MyTM sender, EAPCommandEventArgs e)
    {
      switch (e.CommandName)
      {
        case SPECIAL_COMMAND:
          DoTheSpecialCommand(sender);
          break;
      }
    }

    private void AddMapperCommand(IMapper sender)
    {
      MapperCommand cmd = new MapperCommand();
      //--- Put the command on the MoreMenu (Action Menu) or Toolbar
      MapperCommandLocation location = MapperCommandLocation.MoreMenu;
      location = MapperCommandLocation.ToolBar;
      cmd.Location = location;

      //--- Get the captiona and tooltip from localized text associated with the extension
      string caption = "This is a special command";
      string tooltip = " This does something really special ";
      cmd.Caption = this.TextItems.GetText(COMMAND_CAPTION, caption);
      cmd.Tooltip = this.TextItems.GetText(COMMAND_TOOLTIP, tooltip);

      //--- See MapperCommandAttributes for more details
      //--- This example, forces users to have to select at least one item in list view
      //--- and only be visible on an existing detail, not a new detail
      cmd.Attributes |= UICommandAttrs.RequireSelection | UICommandAttrs.ExistingOnly;
      cmd.WindowOptions = EAPUtil.WindowOpenFeatures(WindowOpenOptions.Standard, 0, 0);

      cmd.ImageURL = "images/special.bmp";
      cmd.Image32URL = "images/special32.bmp";

      sender.Mapper.Commands.Add(SPECIAL_COMMAND, cmd);
    }
  }
}
```

Adding Menu Commands Programmatically

Having explained all that, there is a shortcut method available for when you have derived your Mapper Extension from the CompanyName.Extensions.TypedExtensionBase.

On that class there is a wrapper function to add a MapperCommand either directly through AddToolbarCommand, or a method, CreateCommand, that constructs a command and lets you add the command manually,.

The helper methods take a command id string parameter, This string parameter is used when interrogating the text items collection for text to extract and assign to the Command object. The Command Id is added to one of four suffix strings to extract the necessary data. If there is no image file name in the localized text, then the command is automatically added to the More Menu (Action Menu). When you do have an image, the command is added to the toolbar.

caption = extensionText.GetText(id + "_CAPTION");
image = extensionText.GetText(id + "_IMAGE");
tooltip = extensionText.GetText(id + "_TOOLTIP");
confirmMsg = extensionText.GetText(id + "_CONFIRM_MSG");

```
using NetQuarry.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
  {
    //--- Declare a const name for your command and then you can refer to it in many places
    //--- Like the MapperCommand event handler!
    private const string SPECIAL_COMMAND = "SPECIAL_COMMAND";

    public override void MapperBeforeLayout(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
    {
      UICommandAttrs attrs = UICommandAttrs.RequireSelection | UICommandAttrs.ExistingOnly;
      this.AddToolbarCommand(this.TextItems, sender.Mapper, SPECIAL_COMMAND, attrs);
      //--- Using the other provided method, CreateCommand, you add the command manually
      //--- but you also get a chance to play with the cmd object before it's added.
      MapperCommand cmd = this.CreateCommand(this.TextItems, sender.Mapper, SPECIAL_COMMAND, attrs);
      Sender.Mapper.Commands.Add(SPECIAL_COMMAND, cmd);
    }

    public override void MapperCommand(CompanyName.Data.MyTM sender, EAPCommandEventArgs e)
    {
      switch (e.CommandName)
      {
        case SPECIAL_COMMAND:
          DoTheSpecialCommand(sender);
          break;
      }
    }
  }
}
```

Creating and Handling PDF files

There is a File converter service built into the application that gives you the ability to convert a number of different basic formats into a text file, image file, or PDF file. This example will show how you can create a PDF and email it to somebody as an attachment. For brevity, this example will take a few shortcuts.

If not already added as a service, go to Services in Studio and add a service named “PDFConverter” that uses the component “NetQuarry.Services.PDFConverter”. Set the service type to “pdfconverter”, Service Category to “File” and mark as a singleton.

This example follows the standard techniques of mapper command handling.

Adding your mapper command programmatically.
If you have an extension based on the CompanyName.Extensions.TypedExtensionBase you can use the helper function to add a mapper command.

When handling your command, you have to choose whether you want to handle one selected row in the list. Multiple selected rows in the list, or no selected rows. No rows selected means you want to process all the rows matching the current filter criteria. For more information about controlling the selection behavior, see the CommandAttributes property of the Menu Commands target.
Your choice of selection rules depends on the requirements of your feature and not whether you add the command via metadata or code.

When you receive a command you have to detect which records (if any) have been selected. Use the SelectedKeys property of the mapper to obtain an ArrayList object of selected key values. If the ArrayList object has no items, then no keys are selected.
If not keys are selected, use the mapper’s current filter. If there are keys selected, create an IN clause of the selected keys. The GetMapperKeyFilter method of the CompanyName.Extensions.TypedExtensionBase class is a good example to copy if your mapper extension does not derive from this generic class.

After determining the appropriate filter, create a new mapper object using that filter. Process the records of the new, filtered mapper object and never the mapper passed into event handler.

After performing your command’s actions, you should decide whether other extensions should also attempt to handle your command action. In the default case, the command will be fired on all the extensions linked to the mapper.

You may have set up your extension handler with the highest priority and do not want other extensions to handle any mapper commands after yours. In that case you would set the Result property of the EAPCommandEventArgs parameter with the value ExtResults.ContinueNoMoreExt. This means don’t fire this event on any more extensions attached to the mapper.

Creating and Handling PDF files

```
using NetQuarry.Data;
using CompanyName.Data;
namespace CompanyName.Extensions
{
  public class MyClass : CompanyName.Extensions.TypedMapperBase<CompanyName.Data.MyTypedMapper>
  {
    private const string SEND_PDF = "SEND_PDF";

    public override void MapperCommand(CompanyName.Data.MyTM sender, EAPCommandEventArgs e)
    {
      switch (e.CommandName)
      {
        case SEND_PDF:
          SendPDFFile(sender);
          break;
      }
    }

    private void SendPDFFile(MyTM sender)
    {
      NetQuarry.Template tmpl = appCxt.Templates["send_info"];
      if(tmpl == null) goto theExit;
      NameValueCollection nv = null;
      ArrayList arrKeys = sender.Mapper.SelectedKeys();
      string rowFilter = (arrKeys.Count > 0) ? base.GetMapperKeyFilter(arrKeys, "people_id") :
                  this.Mapper.Filters.GetFilter(GetFilterFlags.IncludeAllTypes);

      using (MyTM myTM = MyTM.OpenReader(this.Application, rowFilter, 0, MapperAttrs.NoRowRequery))
      {
        if(myTM.MoveNext())//--- This might be a while loop.  It depends on what you are doing
        {
          nv = myTM.Mapper.Exec(MapperExecCmds.KeyDisplayCollectionGet, 0) as NameValueCollection;
        }
        myTM.Close();
      }

      string sEmailBody = tmpl.Replace(nv, ContentResolution.High);
      DocumentOptions docOpts = new DocumentOptions("SomeName", sender.description, PageSize.Letter,
                                                    PageOrientation.Portrait, CompressionLevel.Normal,
                                                    50, 45, 25, 20, false, false, false, true, true, true);

      docOpts.Subject = sender.description;
      docOpts.Selectable = true; //--- Selectable text looks MUCH better

      NetQuarry.IFileConverter fc = null;
      fc = (IFileConverter)this.Application.Services.GetServiceInstance("PDFConverter");
      if(fc == null) goto theExit;

      System.Net.Mail.Attachment  attach1 = null;
      System.IO.Stream strm1 = null;

      fc.Convert(sEmailBody, out strm1, ConversionSourceType.HTML, ConversionOutputType.PDF, docOpts);
      attach1 = new System.Net.Mail.Attachment(strm1, "send_me_info.pdf", "application/pdf");

      IEmailService mail = (IEmailService)this.Application.Services.GetServiceInstance ("SmtpMail");

      CompanyName.Common.Session cs = sender.Session;

      mail.Send("system@company.com", cs.DisplayEmail, null, null, "Your PDF","This is the PDF you requested",
                false, new System.Net.Mail.Attachment[]{attach});

theExit:
    }
  }
}
```

Adding and Using Text that can be Localized

The NetQuarry platform gives you the ability to add your own string resources into meta data. This is important because it allows these string resources to be localized into different languages and then selected from the database when needed in the correct language.

The benefits of adding localized strings into the database are that you don't have to recompile any code like you would if you were using a regular resource fie. Your code that takes advantage of text string support is immediately ready to be used for other locales. The text strings are easy to extract and re-import for translation

In the Studio, you add strings to the User Defined Text subform of any object supporting user defined text. Objects supporting user defined text are
Applications, Components (Extensions), Pages, Mappers,

To access the collection of text items on any of these objects, you can refer to the TextItems collection property.

```
using NetQuarry.Data;
using CompanyName.Data;
using CompanyName.Common;
namespace CompanyName.Extensions
{
  public class MyClass : NetQuarry.Data.TypedMapperExtension<CompanyName.Data.MyTypedMapper>
  {
    private const string COMMAND_CAPTION = "COMMAND_CAPTION";
    private const string COMMAND_TOOLTIP = "COMMAND_TOOLTIP";
    private const string STANDARD_ERROR_FMT = "STANDARD_ERROR_FMT";

    public override void RowCurrent(CompanyName.Data.MyTypedMapper sender, EAPEventArgs e)
    {
      //--- Get the caption from the Extension's, text collection
      string caption = "This is a special command";
      caption = this.TextItems.GetText(COMMAND_CAPTION, caption);

      //--- Get the tooltip from the Mapper's, text collection
      string tooltip = "This does something really special";
      tooltip = sender.Mapper.TextItems.GetText(COMMAND_TOOLTIP, tooltip);

      //--- Get a string from the application's text collection
      string errorFmt = "There's been an error in {0}. Don't try that again.";
      //--- Accessing the collection in this way, you might get a null object returned
      //--- The above methods are preferred because you can provide a default
      //--- and if not exists, returns an empty string if no default provided
      errorFmt = sender.Application.TextItems[STANDARD_ERROR_FMT];
     }
    }
  }
}
```

Adding a multi-select list box that provides a find capability

You would use this type of approach to associate many related items to a parent. Traditionally, you could simply add a subform to the parent where the subform has a Multi-Add capability. The multi-add subform approach has the benefit of functioning completely with only a small amount of meta data configuration. Adding the multi-select list box to effectively perform the same task is more complicated and requires a combination of meta data, server side code and client side JavaScript.

The typical use for this multi-select list box is when you would like to perform an association of a parent to multiple related records in a wizard process. So here, we're going to have an example where we create a new person, associated with a company and in the wizard, associate one or more people as team members.

The first job is to set up the required meta data. The basic setup is to have a people page with a subform. The subform contains a tab for displaying the selected team members.

Then there is a wizard page created to create the person and add one or more related team members. The wizard will have two pages. Page 1 for basic person information, company, name and the second page for the related team members.

The wizard is split into two pages to facilitate the team member selection process. Once the company is associated with the person on page 1, then, and only then can we identify the company from which to pull related team members on page 2.

The person wizard page is using the same mapper as the basic person detail, and the team member chooser for the wizard will be flavored for inclusion only on the person wizard.

Assuming the basic person mapper is set up, as well as the basic page structure and navigation relationships…

Adding a multi-select list box that provides a find capability

Add the team_member selection fields on the mapper for the wizard.

Field	Specification
team_member_lst (This is where your selections will be displayed to the user)	Cell Type – ListBox Attributes - ExcludeFromSelect Include Flavor – Wizard Table - + Caption – Team Members CellTypeAttributes – MultiSelect ValidationScript – NewItemValidate
team_member_vals (This is where your selected id's will be stored on the page/mapper in a semi-colon delimited list)	Cell Type – TextBox Attributes - ExcludeFromSelect Include Flavor – Wizard Table - +
team_member_disps (This is where your selected names will be stored on the page/mapper in a semi-colon delimited list)	Cell Type – TextBox Attributes - ExcludeFromSelect Include Flavor – Wizard Table - +
team_member_find (This is the control that provides the find capability. There is an onchange handler specified when a selected item ID is written to this field, the on change handler adds the ID to the team_member_vals field)	Cell Type – Find Attributes - ExcludeFromSelect Include Flavor – Wizard Table - + Caption – Team Members CellTypeAttributes – CascadingCombos, HideTextBox, NoLabelLink, RepeatedSelect OnChange – WizardNewAddVal(event, this, 'team_member'); FindFields – team_member_find=person_id;team_member_find_display=person_name FindFilter – company_id=['company_id] AND display=1 FindMOP – person!detail
team_member_find_display (This is the control that receives the selected display value from the find results. There is an onchange handler specified. When the display value is written to this field, the onchange handler adds the display value to the team_member_lst field)	Cell Type – TextBox Attributes - ExcludeFromSelect Include Flavor – Wizard Table - + OnChange – WizardNewAddLst(event, this, 'team_member');

Adding a multi-select list box that provides a find capability

The next step is to add some client side JavaScript to support the multi select. In your company specific JavaScript file, add the following code. This code is designed to be generic so you can re-use the functionality for any multi-select requirements. This specific code will work for situations where you need to support multi-select items that store an ID value but display a readable value. This is akin to a picklist that is the "limit-to-list" style.

```
// Call this function if you want to make the multi-select field required
function NewItemValidate(fld, val, full, cap)
{
  return (fld.options.length > 0) ? '' : 'You must provide one or more ' + cap + ' you wanka!';
}
// In the wizard, delete all selected items from the item list.
function WizardNewDelItem(evt, btn, ctl_base)
{
  var lst = GetDetailSibling(btn, ctl_base + '_lst');
  var vals = GetDetailSibling(btn, ctl_base + '_vals');
  var disps = GetDetailSibling(btn, ctl_base + '_disps');

  var ii;
  var cnt = lst.options.length;
  var dispItems = '';
  var valItems = '';

  var valArr = (vals != null) ? vals.value.split(';') : null;

  for (ii=cnt-1; ii>=0; ii--)
  {
    if (lst.options[ii].selected)
      lst.remove(ii);
    else
    {
      if (disps != null)
      {
        dispItems = dispItems.length > 0 ? dispItems + ';' + lst.options[ii].text : lst.options[ii].text;
      }
      if (vals != null)
      {
        valItems = valItems.length > 0 ? valItems + ';' + valArr[ii] : valArr[ii];
      }
    }
  }

  if (vals != null)
  {
    vals.value = valItems;
  }
  if (disps != null)
  {
    disps.value = dispItems;
  }
}
//--- On a page, add  newly selected items to the item list.
//--- Use this method as an onchange handler for an "ID" field for list requiring ID and Display
//--- e.g., a "limit to list" picklist
function WizardNewAddVal(evt, ctl, ctl_base)
{
  var vals = GetDetailSibling(ctl, ctl_base + '_vals');
  var item = ctl.value;

  if (item == null || item == '')
  {
    WizardNewDelItem(evt, ctl, ctl_base);
  }
  else
  {
    vals.value = vals.value.length > 0 ? vals.value + ';' + item : item;
  }
}
//--- On a page, add  newly selected items to the item list.
//--- Use this method as an onchange handler for an "DISPLAY" field for list requiring ID and Display
//--- e.g., a "limit to list" picklist
function WizardNewAddLst(evt, ctl, ctl_base)
{
  var lst = GetDetailSibling(ctl, ctl_base + '_lst');
  var disps = GetDetailSibling(ctl, ctl_base + '_disps');
  var item = ctl.value;

  if (item == null || item == '')
  {
    WizardNewDelItem(evt, ctl, ctl_base);
  }
  else
  {
    if (disps != null)
    {
        disps.value = disps.value.length > 0 ? disps.value + ';' + item : item;
    }
    AddOption(lst, item, item);
  }
}
```

Adding a multi-select list box that provides a find capability

The next step is recommended in wizards to correctly support the back/next button functionality. If you use this control on a wizard and don't add this code, then if you move back to a previous page and then forward to the page with the list select, the values selected on the control would be lost (even though the actual underlying values in the "vals" and "name_display" field would still exist.

In the RowCurrent event handler, you need to specify the following code that is only going to run in the case of wizard processing

```
if (EAPUtil.BitsSet(sender.Mapper.Flavor, Flavors.WizardPage))
{
  IField fldList = sender.Mapper.Fields["team_member_lst"];
  IField fldVals = sender.Mapper.Fields["team_member_vals"];
  IField fldDisp = sender.Mapper.Fields["team_member_disps"];

  if (fldList != null && fldVals != null && fldDisp != null)
  {
    SetupMultiList(sender.Database, fldDisp, fldVals, "person_id", "person_name", "person_id",
                   sender.person_id, fldList.BaseControl as System.Web.UI.WebControls.ListBox,
                                                          "person_team_view", false);
  }
}
```

Adding a multi-select list box that provides a find capability

In your utility class, add this code

```
#region SetupMultiList
/// <summary>
///
/// </summary>
/// <param name="db">The database.</param>
/// <param name="fldDisp">The field containing the display names in the list.</param>
/// <param name="fldVals">The field containing the ID's in the list. (may be null)</param>
/// <param name="valsIDField">The field to be used for identifying the related id's - e.g. person_id</param>
/// <param name="valsDisplayfield">The field to be used for identifying the related names - e.g. person_name</param>
/// <param name="parentIDField">The ID field of the owner</param>
/// <param name="parentID">The ID of the owner</param>
/// <param name="lst">The ListBox control used to present the list of current items.</param>
/// <param name="tableName">The table to get access to the people from</param>
/// <param name="display">Whether or not the table has a display paramater</param>
/// <returns>The string ID representing the first item in the list</returns>
public static string SetupMultiList(IDatabase db, IField fldDisp, IField fldVals, string valsIDField, string valsDisplayField,
                                    string parentIDField, string parentID, ListBox lst, string tableName, bool display)
{
  //--- variable to return the first item in the list
  string sFirstItem = null;
  string sItemVal = null;
  string sItemDisp = null;

  //--- Clear out the list.  We refill w/ every round trip.
  if (lst != null) lst.Items.Clear();

  //--- Get the real list of codes.
  string sVals = fldVals != null ? EAPUtil.ToString(fldVals.Value) : null;
  string sDisps = EAPUtil.ToString(fldDisp.Value);

  //--- Fill the list either from the already know list, or initialized for the owner.
  if (fldDisp.Dirty)
  {
    //--- The Disp field is dirty meaning we've already set it and it now contains the real set of items.  Fill the list from it.
    string[] arrVals = (fldVals != null) ? sVals.Split(';') : null;
    string[] arrDisps = sDisps.Split(';');
    for (int ii = 0; ii < arrDisps.Length; ii++)
    {
      if (fldVals != null) sItemVal = arrVals[ii];
      sItemDisp = arrDisps[ii];
      if (!string.IsNullOrEmpty(sItemDisp))
      {
        if (lst != null)
        {
          lst.Items.Add(sItemDisp);
        }
        if (string.IsNullOrEmpty(sFirstItem))
        {
          sFirstItem = (fldVals != null) ? sItemVal : sItemDisp;
          if (lst == null) break;
        }
      }
    }
  }
  else if (!string.IsNullOrEmpty(parentID))
  {
    //--- We need to initialize the list according to the owner.
    string sSql = string.Format(@"SELECT DISTINCT {0}, {1} FROM {2} WHERE {3} = {4} {5} ORDER BY 2",
                  valsIDField, valsDisplayField, tableName, parentIDField, EAPUtil.AnsiQuote(parentID),
                  (display == true ? "AND display = 1" : string.Empty));

    DataTable tbl = db.OpenTable(sSql, "SetupMultiList");
    foreach (DataRow row in tbl.Rows)
    {
      sItemVal = EAPUtil.ToString(row[0]);
      sItemDisp = EAPUtil.ToString(row[1]);

      if (!string.IsNullOrEmpty(sItemVal))
      {
        sVals += sVals.Length > 0 ? ";" + sItemVal : sItemVal;
        sDisps += sDisps.Length > 0 ?  ";" + sItemDisp: sItemDisp;
        if (lst != null)
        {
          lst.Items.Add(sItemDisp);
          if (lst.Items.Count >= basOrders.Constants.MAX_COST_CODE_ROWS) break;
        }
        if (string.IsNullOrEmpty(sFirstItem))
        {
          sFirstItem = (fldVals != null) ? sItemVal : sItemDisp;
        }
      }
    }
    //--- And set the real list of items.
    if (fldVals != null) fldVals.Value = sVals;
    fldDisp.Value = sDisps;
  }
          Exit:
          return (sFirstItem);
}
#endregion
```

Adding a multi-select list box that provides a find capability

If you are requiring a multi-select list where the ID and display are the same value. That is, like a simple picklist, then you don't need to create the "_vals" field, or the "_find_display" field.

Field	Specification
team_member_lst (This is where your selections will be displayed to the user)	Cell Type – ListBox Attributes - ExcludeFromSelect Include Flavor – Wizard Table - + Caption – Team Members CellTypeAttributes – MultiSelect ValidationScript – NewItemValidate
team_member_disps (This is where your selected names will be stored on the page/mapper in a semi-colon delimited list)	Cell Type – TextBox Attributes - ExcludeFromSelect Include Flavor – Wizard Table - +
team_member_find (This is the control that provides the find capability. There is an onchange handler specified when a selected item ID is written to this field, the on change handler adds the ID to the team_member_vals field)	Cell Type – Find Attributes - ExcludeFromSelect Include Flavor – Wizard Table - + Caption – Team Members CellTypeAttributes – CascadingCombos, HideTextBox, NoLabelLink, RepeatedSelect OnChange – WizardNewAddLst(event, this, 'team_member'); FindFields – team_member_find=person_id FindFilter – company_id=['company_id] AND display=1 FindMOP – person!detail

Your RowCurrent handler code would pass null as the fldVals object

```
if (EAPUtil.BitsSet(sender.Mapper.Flavor, Flavors.WizardPage))
{
  IField fldList = sender.Mapper.Fields["team_member_lst"];
  IField fldDisp = sender.Mapper.Fields["team_member_disps"];

  if (fldList != null && fldDisp != null)
  {
    SetupMultiList(sender.Database, fldDisp, null, "person_id", "person_name", "person_id",
                   sender.person_id, fldList.BaseControl as System.Web.UI.WebControls.ListBox,
                                                            "person_team_view", false);
  }
}
```